



INVESTOR IN PEOPLE

The Patent Office
Concept House
Cardiff Road
Newport
South Wales
NP10 8QQ



I, the undersigned, being an officer duly authorised in accordance with Section 74(1) and (4) of the Deregulation & Contracting Out Act 1994, to sign and issue certificates on behalf of the Comptroller-General, hereby certify that annexed hereto is a true copy of the documents as originally filed in connection with the patent application identified therein.

In accordance with the Patents (Companies Re-registration) Rules 1982, if a company named in this certificate and any accompanying documents has re-registered under the Companies Act 1980 with the same name as that with which it was registered immediately before re-registration save for the substitution as, or inclusion as, the last part of the name of the words "public limited company" or their equivalents in Welsh, references to the name of the company in this certificate and any accompanying documents shall be treated as references to the name with which it is so re-registered.

In accordance with the rules, the words "public limited company" may be replaced by p.l.c., plc, P.L.C. or PLC.

Re-registration under the Companies Act does not constitute a new legal entity but merely subjects the company to certain additional company law rules.

**CERTIFIED COPY OF
PRIORITY DOCUMENT**

Signed

Dated 22 February 2001

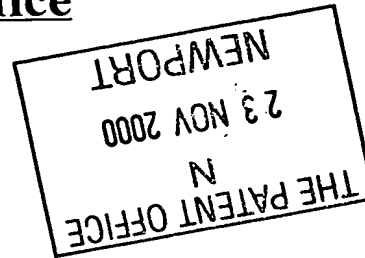
This Page Blank (uspto)

The Patent Office

1/77

Patents Act 1977
Rule 16

23NOV00 E585943-1 000611
P01/7700 0.00-0028516.3



Request for grant of a patent

The Patent Office
Concept House
Cardiff Road
Newport
South Wales NP10 8QQ

1.	Your reference	GB920000099GB1		
2.	Patent application number (The Patent Office will fill in this part)	23 NOV 2000	0028516.3	
3.	Full name, address and postcode of the or of each applicant (underline all surnames)	INTERNATIONAL BUSINESS MACHINES CORPORATION Armonk New York 10504 United States of America		
	Patents ADP number (if you know it)	519637001		
	If the applicant is a corporate body, give the country/state of its incorporation	State of New York United States of America		
4.	Title of the invention	DATA LOGGING METHOD, APPARATUS, SYSTEM AND COMPUTER PROGRAM		
5.	Name of your agent (if you have one)	M J Jennings		
	"Address for Service" in the United Kingdom to which all correspondence should be sent (including the postcode)	IBM United Kingdom Limited Intellectual Property Department Hursley Park Winchester Hampshire SO21 2JN		
	Patents ADP number (if you know it)	7783715001		
6.	If you are declaring priority from one or more earlier patent applications, give the country and the date of filing of the or of each of these earlier applications and (if you know it) the or each application number	Country	Priority App No (if you know it)	Date of filing (day/month/year)
7.	If this application is divided or otherwise derived from an earlier UK application, give the number and the filing date of the earlier application	No of earlier application	Date of filing (day/month/year)	

Is a statement of inventorship and of right to grant of a patent required in support of this request? (Answer 'Yes' if:

Yes

- a) any applicant named in part 3 is not an inventor, or
b) there is an inventor who is not named as an applicant, or
c) any named applicant is a corporate body.)

9. Enter the number of sheets for any of the following items you are filing with this form. Do not count copies of the same document

Continuation sheets of this form

Description	11
Claim(s)	4
Abstract	1
Drawing(s)	9 x 9

10. If you are also filing any of the following, state how many against each item.

Priority documents

Translations of priority documents

Statement of inventorship and right to grant of a patent (Patents Form 7/77)

2 ✓

Request for preliminary examination and search (Patents Form 9/77)

Request for substantive examination (Patents Form 10/77)

Any other documents (please specify)

11.

I/We request the grant of a patent on the basis of this application

Signature



M J Jennings

21 November 2000
Date

12.

Name and daytime telephone number of person to contact in the United Kingdom

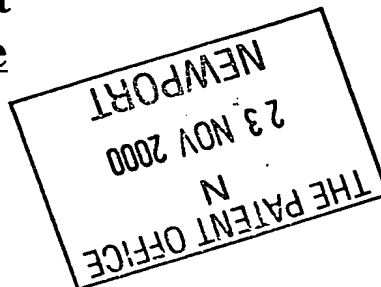
J N Watson
01962 818955

The Patent Office

7/77


Patents Act 1977
Rule 15

Statement of inventorship and of
right to grant of a patent



The Patent Office

Concept House
Cardiff Road
Newport
South Wales NP10 8QQ

1.	Your reference	GB920000099GB1
2.	0028516.3 <i>(if you know it)</i>	23 NOV 2000
3.	Full name of the or of each applicant	INTERNATIONAL BUSINESS MACHINES CORPORATION
4.	Title of invention	DATA LOGGING METHOD, APPARATUS, SYSTEM AND COMPUTER PROGRAM
5.	State how the applicant(s) derived the right from the inventor(s) to be granted a patent	By employment and agreement
6.	How many, if any, additional Patents Forms 7/77 are attached to this form?	
7.	I/We believe that the person(s) named over the page (and on any extra copies of this form) is/are the inventor(s) of the invention which the above patent application relates to.	
		 Signature <u>M J Jennings</u>
		21 November 2000 Date
8.	Name and daytime telephone number of person to contact in the United Kingdom	J N Watson 01962 818955

Enter the full names, addresses and postcodes of the inventors in the boxes and underline the surnames

Andrew John SCHOFIELD
(UK resident)
c/o IBM United Kingdom Limited
Intellectual Property Law
Hursley Park
Winchester
Hampshire
SO21 2JN
UK

Patents ADP number *(if known)*

Patents ADP number *(if known)*

If there are more than three inventors, please write their names and addresses on the back of another Patents Form 7/77 and attach it to this form

REMINDER

Have you signed the form?

Patents ADP number *(if known)*

DATA LOGGING METHOD, APPARATUS, SYSTEM
AND COMPUTER PROGRAM

The present invention relates to the field of data processing systems, and more particularly to the logging of data within such systems.

Many software systems which manage important data record all updates to non-volatile data in what's known as a log. Figure 1 provides an overview of the process according to the prior art. A central server 10 holds some non-volatile data 20. A client 50 runs software 60 which causes some of the data 20 to be updated, for example, via a network connection to the server (not shown). The updated information is first stored in a buffer at the server (not shown) and at a predetermined point in time, the updated data is written out to a log 40. This, along with the buffering, is controlled by a log management component 45. The purpose of the log is to provide a high-integrity history of the data 20 and the operations upon it for recovery purposes. Note, the software for updating data 20 may run as a separate process at the server. Further the log 40 may sit across the network.

The logical view of the data in the log is of a sequence of records. New data is written to the end of the sequence and records are never changed once written. Figure 2a shows an exemplary logical view of a log according to the prior art. In this example, there are three records A, B, C and these occupy one contiguous area of storage which grows from left to right (see time arrow).

The physical view is quite different. The data in the log is split into fixed-size blocks called pages, each of which is typically 4 Kilobytes in size. Records are tightly packed into the pages without any gaps between them. Page boundaries are overlapped where necessary. Figure 2b shows an exemplary physical view according to the prior art. A page may consist of whole record(s) (A), a number of record fragments (B1, B2, C1, C2, C3), or a combination of the two. Record fragments correspond to those records which have been split across page boundaries. For example, it can be seen that C1, C2 and C3 constitute a record C. Note, each log record in the log is uniquely identified by a number called its Log Sequence Number (LSN). An LSN gives the byte offset of the start of a record from the beginning of the log, and so it is a simple matter to calculate the location of a log record given its LSN.

Typically, the last page in the log is not full of data and is known as a partial page 100. When a new record is added to the log, it may fill a partial page with data and cause a new partial page to be added to the end of the log.

While it is true in the logical view of the log that data is only ever appended to the end of the log, in the physical view, this is not necessarily true and existing pages of log data can be overwritten. It is not possible to guarantee that all of the data in a page will be written atomically as a single operation and indivisible in any way. During the overwriting process, it is possible for the page to become corrupted. By way of example, a partial page contains a record fragment C3 as shown in figure 2b. A further write would cause that partial page to be overwritten with the C3 fragment and another record (D) or record fragment (D1) such that in the majority of cases, the page is filled. However, if the power is lost, for example, whilst the page is being overwritten then the page may be written incorrectly such that the C3 fragment is incomplete or not there at all.

It is therefore necessary to ensure that log records are not lost or corrupted unintentionally. One such method is to rely on fault-tolerant hardware, such as, for example, an SSA fast-write cache. This is a cache typically within a disk drive housing the log. When a server's operating system (o/s) issues a write to the disk drive this gets stored in the cache and control is relinquished back to the o/s for further transaction processing. The data can then be written out to disk asynchronously. Caching to volatile memory is disliked since it jeopardises data integrity. This is because the o/s is fooled into believing that a transaction or transaction part has been recorded on disk when in reality this is yet to happen. If volatile memory fails then the data within is lost. An SSA fast-write cache however is battery backed (i.e. a power failure would not be a problem, since the batteries would simply take over). This kind of system is however extremely expensive and not cost effective in most commercial environments.

Another method is illustrated by figure 3. It can be seen that each write starts a new page. Hence there is no chance of corrupting the most recent version of a page. Record A makes for a partial page X at position N. The full version of that page X(i) is written out to position N+1. Page Y at position P again makes for a partial page. A fuller version of that page Y(ii) is written to position P+1 and the complete version Y(iii) is

written at position P+2. What should be a single page of log data might therefore actually occupy more than one page of space in the log. This makes reading the log inefficient and it also makes it harder to manage the space requirements of the log. Housekeeping functions activated by the log management component typically determine that a particular page is redundant and available for overwriting (e.g. old versions of page Y(iii)). Note, the complete versions of pages may also be available for overwriting (i.e. if a checkpoint has been taken which includes those pages). However this method does mean that the log fills up more quickly and partial pages will be mixed in with full pages, both of which are undesirable. Further, it is not possible to predict the position of a particular log record via its LSN. Instead it would be necessary to scan the log each time a record was read or to hold some kind of directory information about the log records to locate a particular record. This is undesirable.

A careful-writing scheme or algorithm is therefore typically adopted to ensure that data which has already been written to the log is not corrupted by subsequent writes. The ping-pong algorithm is one example of such a scheme and is used, for example, by DB2 Universal Database, available from IBM Corporation. In this scheme, the last page of log data may actually appear twice in the log, once in the expected position (position N) and once again in the next location in the log (position N+1). Only the last page in the log can appear twice like this in what is known as a ping-pong pair. When a page of log data is superseded by a newer version, the previous version is not overwritten directly because that may result in data corruption. Instead, subsequent versions alternate in their locations between position N and position N+1 ensuring that the previous, most recent version is not corrupted when the newer version is written. When the page is being read, it is necessary to check both in location N and location N+1. Either version of the page may be newer depending on how many times the page has been updated. When the page is finally filled, it is written to its own proper location N before the first version of the next page of the log can be written into the next location N+1. This scheme gives the advantage that you can calculate the location of a log record by its LSN.

Although data corruption is, in the main, prevented, a major disadvantage of the ping-pong scheme is efficiency. In this scheme, along with the other methods mentioned above, data is written directly to the log without caching by the central server's operating system or hardware (not shown in figure 1). So, when the log management component issues a request

to the operating system to write data to the log, control is not returned to the log management component until the data is stably recorded in a non-volatile medium. This is necessary since the server needs to know what has been written to the disk in order to ensure data integrity (see above). This also means that it is time-consuming to write data to the log because the cost of waiting for the relatively slow disk drive cannot be amortised across many write operations by use of a cache. Luckily, when most records are written to the log, they do not have to be recorded until a point of consistency such as the completion of a unit of work. However, the cost of writing to the log is still a major factor in the performance of many data systems. In striving to avoid the consequences of data corruption, the ping-pong scheme requires that when a page is first written as a partial, the page is typically written three times before the full version is in place. This is because it is most common for only one partial page to be written.

Figure 4 illustrates the operation of the ping-pong scheme and the performance problems associated with it. (Note, the pages with the thick borders indicate the data actually written by each disk write - this is also so in figures 5, 7b and 7c.) As previously mentioned, it is most common for the last page in a log to be written as a partial page and for it to be filled via subsequent writes. Typically, the page is full by the time the second write has been completed. Thus in the figure, partial page 200 represents the "before" scenario. A record fragment B1 makes for a full page 200(i), which is written (Write 1) to position N+1 such that there is no risk of overwriting the most recent version of the page 200 at position N. Write 2 subsequently copies the full version of the page 200(ii) back to its expected location at position N. A third write writes the next partial page 210 into position N+1. The next page along (i.e. at position N+2 (not shown)) then becomes available for the ping-pong pair for partial page 210.

Logically, it was completely unnecessary to write page 200(i) into position N+1, but this was required to ensure that the latest version of page 200 in position N was not corrupted by a power failure whilst page 200(i) was being written. Similarly, it was not permitted to combine writes 2 and 3 into a single write since it would be possible to corrupt the data in both locations N and N+1. (This would, however, make for a less I/O bound solution.) This is shown in figure 5. It can be seen that write 1 writes the full version of the page 200(i) to position N+1 (as in

figure 4) and write 2 then writes this into position N and page 210 into position N+1.

Data is written out to disk a sector at a time. A disk sector is typically 512 bytes, whereas a page is typically 4 Kilobytes. Thus it is quite possible that data will not be written out from the beginning of a page. Further, disk drives commonly optimise the order of writing to match the position of the disk under the write head at the time that the request arrives in the disk drive. Since the sectors which have to be written to the disk drive are not therefore necessarily written in the order which one might expect, it is possible that only part of page 210 is written, followed by part of page 200(ii), followed by a power failure. If this happens then both page 200 and page 200(i) are corrupted and records lost. This will be further explained with reference to figure 6 below.

In aiming to write the full version of page 200(ii) and partial page 210 in one write (write 2), the problem outlined above occurs. The write head begins writing at location R and continues up to location S. Note, location R is not at the beginning of partial page 210. Writing then starts again at location T and ends abruptly at location U due to, for example, a power failure. The logical view of the end of the log is as if page 200(ii) does not exist because both copies (200(i) and 200(ii)) of that page are either overwritten or corrupt. The data in location N contains a corrupted version 202 of page 200(ii). Location N+1 contains a corrupt page 205 consisting of some of page 200(ii) and some of page 210. There is no longer a good copy of page 200(ii) and so no way of recovering the information.

Accordingly, the invention provides a method for logging updates to a plurality of data records, wherein said updates are logged to an area of non-volatile storage and said storage area is divided into a plurality of discrete pages, wherein a page partially full of data is known as a partial page, and said data relates to at least one update to said plurality of data records, said method comprising the steps of: writing, via a single write operation, a first partial page to said storage area both at position N and position N+1; determining whether a subsequent write operation will fill said first partial page; and responsive to determining that said subsequent write will fill said first partial page, writing a full version of said page to position N, leaving N+1 unchanged.

According to the preferred embodiment, position N designates the next available page in non-volatile storage. This will either be an, as yet, unfilled page in storage or a redundant page (i.e. available to be overwritten). As previously mentioned, housekeeping functions preferably determine which pages are to be classed as redundant. Preferably position N+1 designates the next available page after position N. Note, in an alternative embodiment, positions N and N+1 simply represent suitable positions within the non-volatile storage. Such suitable positions are not necessarily the next ones available. Further preferably an update comprises any one of: adding a new data record; deleting a data record; modifying a data record.

Preferably if it is determined that the subsequent write will not fill the first partial page, a fuller version of the first partial page is written to position N+1, leaving N unchanged. Data is henceforth written alternately to the pages at positions N and N+1, such that the most recent version of the page is not overwritten, until one of the page at position N and the page at position N+1 is full of data. According to the preferred embodiment, this is to ensure that the most recent version of the page can not be corrupted whilst it is being overwritten with a newer version. Such corruption could occur if, for example, a power failure occurred in the midst of the overwriting process.

According to a preferred embodiment, responsive to determining that the page at position N+1 is full of data, this page of data is preferably copied to position N. Once it is determined that the page at position N is full of data, a second partial page is, according to one embodiment, written, via a single write operation, to the storage area both at position N+1 and position N+2. Thus the page at position N+1 is overwritten by the second partial page. It is safe to do this since the data at position N+1 is a duplicate of that at position N. Note, position N+1 now preferably becomes a new position N and likewise, position N+2 becomes a new position N+1.

According to another embodiment, responsive to determining that the page at position N is full of data, a full page of new data is written to position N+1 such that the first partial page at position N+1 is overwritten by the full page of data.

It is possible to write a partial page and duplicate it, via a single write operation, because the page can preferably be duplicated in a buffer

which is then flushed to the non-volatile storage. This is extremely advantageous since the cost of this duplication is very small compared with the cost of a disk write.

5 As previously mentioned, the cost of logging is a major factor in the performance of many data systems. Thus according to the preferred embodiment, the present invention provides an optimised ping-pong algorithm. The invention preferably trades the cost of writing a duplicate copy of the current page against the cost of an extra disk write in the
10 common case where the page is written once as a partial page (to positions N and N+1) and then the next time as a full page (to position N). This is a common occurrence and so the optimisation is an important one. For all other situations, the number of writes required to create a full version of a page is preferably the same as for the unoptimised ping-pong algorithm.

15 Preferably therefore in cases where more than two partial versions of the same page are required, two disk writes are required to extend the log by a page if an even number of partial versions were written and three disk writes if an odd number of partial versions were written (the last partial
20 will preferably have to be written in the page's own location thus necessitating the third write). It is unusual to write this many partial versions of the last page in the log, and so this is not a problem in practice.

25 Examples of software systems which could use a log include relational databases, reliable messaging systems (e.g. MQSeries available from IBM Corporation) and component transactions servers. For example, in a messaging system which provides once-and-once-only delivery semantics for messages sent between applications communicating across a network via the
30 messaging system, the message data and control information for reliable message transmission could be recorded in a log. In such a system, performance tests indicate, according to the preferred embodiment, up to 20% increase in throughput for a simulation of a typical workload attributed to the optimisation disclosed. (Note, in MQSeries for typical
35 workloads it is rare for more than two partial versions of a particular page to be written before the page is filled.)

40 An additional advantage is that this optimisation does not change the format of the log data or any logic not concerned directly with writing log records. It is therefore still possible, according to the preferred embodiment, to calculate the location of a log record given its LSN.

According to another aspect, the invention provides a computer program product comprising computer program code stored on a computer readable storage medium which, when executed on a computer, performs the method described above.

5 According to yet another aspect, the invention provides an apparatus for logging updates to a plurality of data records, wherein said updates are logged to an area of non-volatile storage and said storage area is divided into a plurality of discrete pages, wherein a page partially full
10 of data is known as a partial page, and said data relates to at least one update to said plurality of data records, said apparatus comprising: means for writing, via a single write operation, a first partial page to said storage area both at position N and position N+1; means for determining whether a subsequent write operation will fill said first partial page; and
15 means responsive to determining that said subsequent write will fill said first partial page, writing a full version of said page to position N, leaving N+1 unchanged.

20 According to another aspect, the invention provides a system for logging updates to a plurality of data records held at a server, wherein said updates are received from a client and are logged by said server to an area of non-volatile storage, said storage area being divided into a plurality of discrete pages, wherein a page partially full of data is known as a partial page, and said data relates to at least one update to said
25 plurality of data records, said system comprising: means for receiving said updates from a client; means for writing, via a single write operation, a first partial page to said storage area both at position N and position N+1; means for determining whether a subsequent write operation will fill said first partial page; and means responsive to determining that
30 said subsequent write will fill said first partial page, writing a full version of said page to position N, leaving N+1 unchanged.

35 A preferred embodiment of the present invention will now be described, by way of example only, and with reference to the following drawings:

Figure 1 show an overview of logging according to the prior art;

Figure 2a shows the logical view of the records in a log;

40 Figure 2b shows the physical view of the records in a log;

Figure 3 illustrates a method of ensuring that pages in a log are not unintentionally lost or corrupted;

Figure 4 illustrates the ping-pong careful writing scheme;

Figure 5 shows a variation on the ping-pong careful writing scheme, where writes two and three of figure 4 are combined into a single write;

Figure 6 shows the problem associated with non-sequential writing to disk; and

Figures 7a to 7c illustrate the operation of the present invention according to a preferred embodiment.

According to a preferred embodiment, the present invention provides an optimised ping-pong algorithm. The modified algorithm optimises the case where only one partial version of a page is written. For all other situations the number of writes required to create a full version of a page is preferably the same as for the unoptimised ping-pong algorithm. (In MQSeries for typical workloads it is rare for more than two partial versions of a particular page to be written before the page is filled.)

Figure 7a is a flowchart of the operation of the present invention according to a preferred embodiment. It should be read in conjunction with figures 7b which shows the format of log records written out in accordance with a first pass of the flowchart of figure 7a. In this example, a partial page is filled at a second attempt.

At step 400 the first version of partial page 500 is written twice to the end of the log, once in location N and once in location N+1 (the "before" scenario). This requires that the page is duplicated in the buffer from which the data is written to disk, thus allowing two copies to be written with a single disk write. The cost of this duplication is very small compared with the cost of a disk write. The choice of which of the copies of the partial page is overwritten next is made based on whether the page is being overwritten by another partial version of the page or a full version of the page (step 420). If it is a full version, the full version 500(i) is written (write 1) into position N (step 430). This is in the correct final location for the full page of data and means that the next page or partial page 510 of log data can be written to position N+1 in a second disk write (write 2). Of course if its another partial page (as in

figure 7b) then the page is also duplicated to position $N+2$ (step 410). The preferred embodiment has thus saved a costly extra disk write.

5 Next, consider the case where the first partial version of a page is overwritten by another partial version of the page. This is shown in Figure 7c. Two copies of partial page 600 are written at step 400. The test at step 420 yields no. Thus partial page 600(i) is written to position $N+1$ at step 440 (write 1). Write 1 then becomes the "before" scenario since a write only counts if it is going to make for a full page. At step 450, it is determined that a subsequent write will fill the page. 10 Therefore, the full version is written 600(ii) is written to position N (write 1) at step 430. The flow chart then loops round again (step 400) to write out two copies of the next partial page 610 at positions $N+1$ and $N+2$ (that is assuming that it is a partial page that is to be written). 15 According to the preferred embodiment, the number of writes is the same as for the unoptimised version of the ping-pong algorithm.

20 If at step 450 it is determined that the page will still not be filled then a fuller version of the partial page is written to position N (step 460). If the test at step 470 yields no then the flowchart loops round to step 440 and a further fuller version of the partial page is written to position $N+1$. Otherwise a full version of the page is written to position $N+1$ at step 480 and then copied to position N at step 430. The flowchart then loops round again to step 400 assuming that another partial 25 page is to be written.

30 Thus it can be seen that in cases where more than two partial versions of the same page are required, two disk writes are preferably required to extend the log by a page if an even number of partial versions were written and three disk writes if an odd number of partial versions were written (the last partial will preferably have to be written in the page's own location thus necessitating the third write). It is unusual to write this many partial versions of the last page in the log, and so this is not a problem in practice. In any case, the number of writes is the 35 same as for the unoptimised ping-pong algorithm.

40 According to the preferred embodiment, the invention trades the cost of writing a duplicate copy of the last page of the log against the cost of a third disk write in the common case where the last page of the log is written once as a partial page and then the next time as a full page. It is common for only one version of a partial page to be written, before the

page is overwritten with a full version and so the optimisation is an important one. An additional advantage is that this optimisation does not change the format of the log data or any logic not concerned directly with writing log records.

5

Examples of software systems which could use a log include relational databases, reliable messaging systems (e.g. MQSeries available from IBM Corporation) and component transactions servers. For example, in a messaging system which provides once-and-once-only delivery semantics for messages sent between applications communicating across a network via the messaging system, the message data and control information for reliable message transmission could be recorded in a log. In such a system, performance tests indicate, according to a preferred embodiment, up to 20% increase in throughput for a simulation of a typical workload attributed to the optimisation disclosed.

10

15

Throughout the present application writing/adding to the end of the log has been referred to. It is to be understood that the end of the log, may actually constitute the next available location, which could, for example, be at the beginning of the log if the log has been filled and has wrapped round (e.g. in circular logging) or the next suitable position which is not necessarily the next available location. Housekeeping functions preferably determine which pages are available for overwriting. Further, a partial page is not always written. A full page may be written at a first attempt.

20

25

CLAIMS

1. A method for logging updates to a plurality of data records, wherein said updates are logged to an area of non-volatile storage and said storage area is divided into a plurality of discrete pages, wherein a page partially full of data is known as a partial page, and said data relates to at least one update to said plurality of data records, said method comprising the steps of:

writing, via a single write operation, a first partial page to said storage area both at position N and position N+1;

determining whether a subsequent write operation will fill said first partial page; and

responsive to determining that said subsequent write will fill said first partial page, writing a full version of said page to position N, leaving N+1 unchanged.

2. The method of claim 1, wherein position N designates the next available page in non-volatile storage, and wherein position N+1 designates the next available page after position N.

3. The method of claim 1 or 2 comprising the step of:

responsive to determining that said subsequent write will not fill said first partial page, writing a fuller version of said first partial page to position N+1, leaving N unchanged.

4. The method of claim 3 comprising the step of:

alternately writing data to the pages at positions N and N+1, such that the most recent version of the page is not overwritten, until one of the page at position N and the page at position N+1 is full of data.

5. The method of claim 4 comprising the step of:

responsive to determining that the page at position N+1 is full of data, copying said page of data to position N.

6. The method of any preceding claim, comprising the step of:

responsive to determining that the page at position N is full of data, writing, via a single write operation, a second partial page to said storage area both at position N+1 and position N+2 such that the page at position N+1 is overwritten by said second partial page.

5 7. The method of claim 6, wherein position N+1 becomes a new position N, and position N+2 becomes a new position N+1.

10 8. The method of claim any of claims 1 to 5, further comprising the step of:

15 responsive to determining that the page at position N is full of data and responsive to determining that new data corresponds to a full page, writing a full page of new data to position N+1 such that the first partial page at position N+1 is overwritten by said new data.

20 9. The method of any preceding claim, wherein said updates are written to a buffer before being logged to an area of non-volatile storage, and wherein said first partial page is duplicated in said buffer before being written to said storage area at position N and position N+1 via said single write operation.

25 10. A computer program comprising computer program code which, when executed on a computer, performs the method of any of claims 1 to 9.

30 11. Apparatus for logging updates to a plurality of data records, wherein said updates are logged to an area of non-volatile storage and said storage area is divided into a plurality of discrete pages, wherein a page partially full of data is known as a partial page, and said data relates to at least one update to said plurality of data records, said apparatus comprising:

35 means for writing, via a single write operation, a first partial page to said storage area both at position N and position N+1;

means for determining whether a subsequent write operation will fill said first partial page; and

40 means responsive to determining that said subsequent write will fill said first partial page, writing a full version of said page to position N, leaving N+1 unchanged.

12. Apparatus of claim 11, wherein position N designates the next available page in non-volatile storage, and wherein position N+1 designates the next available page after position N.

5 13. Apparatus of claim 11 or 12 comprising:

means, responsive to determining that said subsequent write will not fill said first partial page, for writing a fuller version of said first partial page to position N+1, leaving N unchanged.

10 14. Apparatus of claim 13 comprising:

means for alternately writing data to the pages at positions N and N+1, such that the most recent version of the page is not overwritten, until one of the page at position N and the page at position N+1 is full of data.

15 15. Apparatus of claim 14 comprising:

20 means, responsive to determining that the page at position N+1 is full of data, for copying said page of data to position N.

16. Apparatus of any of claims 11 to 15, comprising:

25 means, responsive to determining that the page at position N is full of data, for writing, via a single write operation, a second partial page to said storage area both at position N+1 and position N+2 such that the page at position N+1 is overwritten by said second partial page.

30 17. The method of claim 16, wherein position N+1 becomes a new position N, and position N+2 becomes a new position N+1.

18. Apparatus of claim any of claims 11 to 15, further comprising:

35 means, responsive to determining that the page at position N is full of data and responsive to determining that new data corresponds to a full page, for writing a full page of new data to position N+1 such that the first partial page at position N+1 is overwritten by said new data.

40 19. Apparatus of any of claims 11 to 18, wherein said updates are written to a buffer before being logged to an area of non-volatile storage, and

wherein said first partial page is duplicated in said buffer before being written to said storage area at position N and position N+1 via said single write operation.

5 20. A system for logging updates to a plurality of data records held at a server, wherein said updates are received from a client and are logged by said server to an area of non-volatile storage, said storage area being divided into a plurality of discrete pages, wherein a page partially full of data is known as a partial page, and said data relates to at least one
10 update to said plurality of data records, said system comprising:

 means for receiving said updates from a client;

 means for writing, via a single write operation, a first partial page
15 to said storage area both at position N and position N+1;

 means for determining whether a subsequent write operation will fill said first partial page; and

20 means responsive to determining that said subsequent write will fill said first partial page, writing a full version of said page to position N, leaving N+1 unchanged.

ABSTRACT

DATA LOGGING METHOD, APPARATUS, SYSTEM
AND COMPUTER PROGRAM

5 The invention relates to logging updates to a plurality of data
records. The updates are logged to an area of non-volatile storage which
is divided into a plurality of discrete pages. A page partially full of
data is known as a partial page and the data relates to at least one update
10 to the plurality of data records. To start with a first partial page is
written, via a single write operation, to the storage area both at position
N and position N+1. (Position N designates the next available location in
non-volatile storage.) Next it is determined whether a subsequent write
operation will fill the first partial page and responsive to determining
15 that the subsequent write will indeed fill the first partial page, a full
version of the page is written to position N, leaving N+1 unchanged.

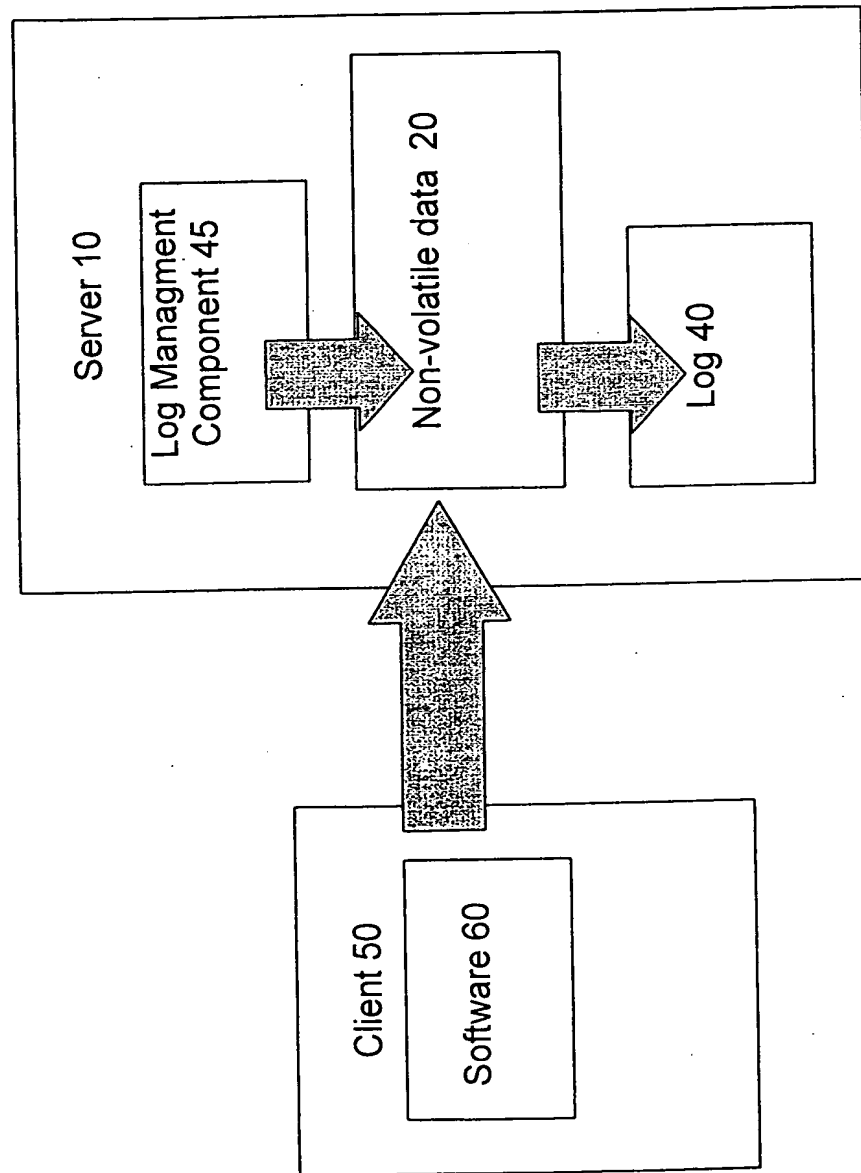


Figure 1

This Page Blank (uspto)

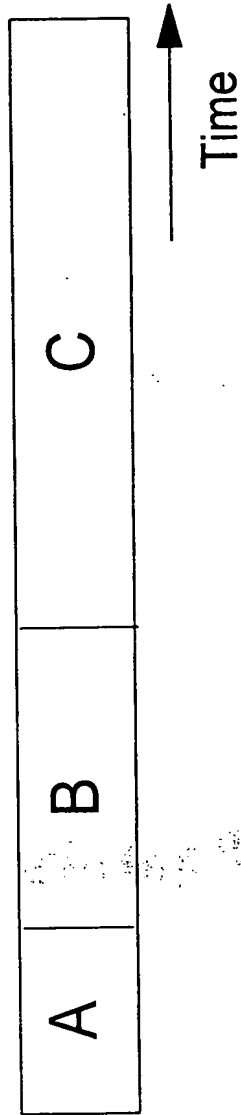


Figure 2a

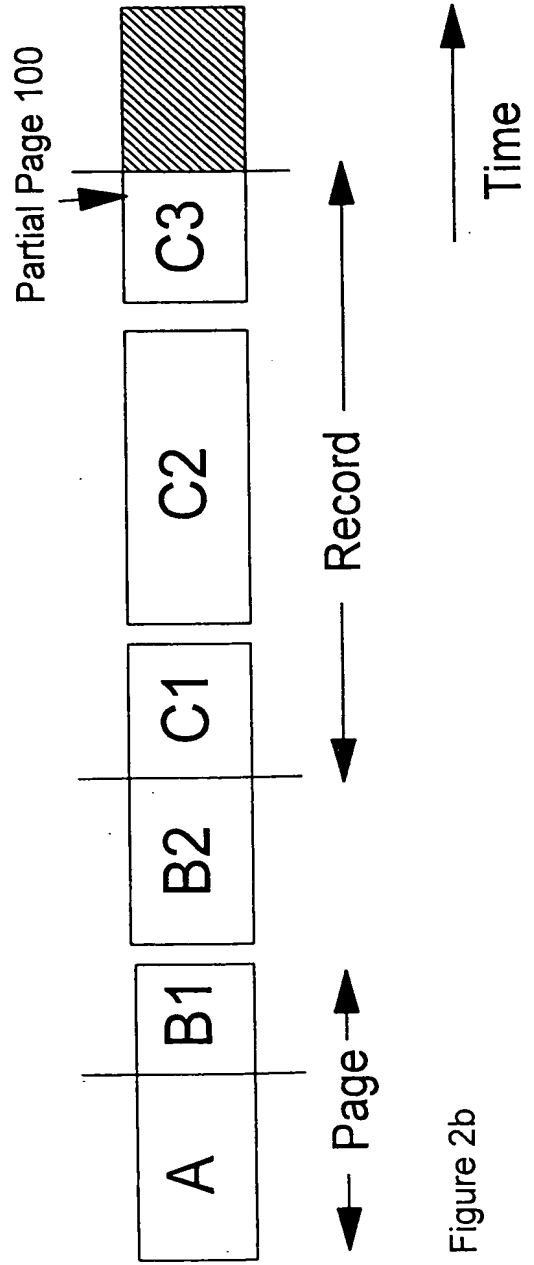


Figure 2b

This Page Blank (uspio)

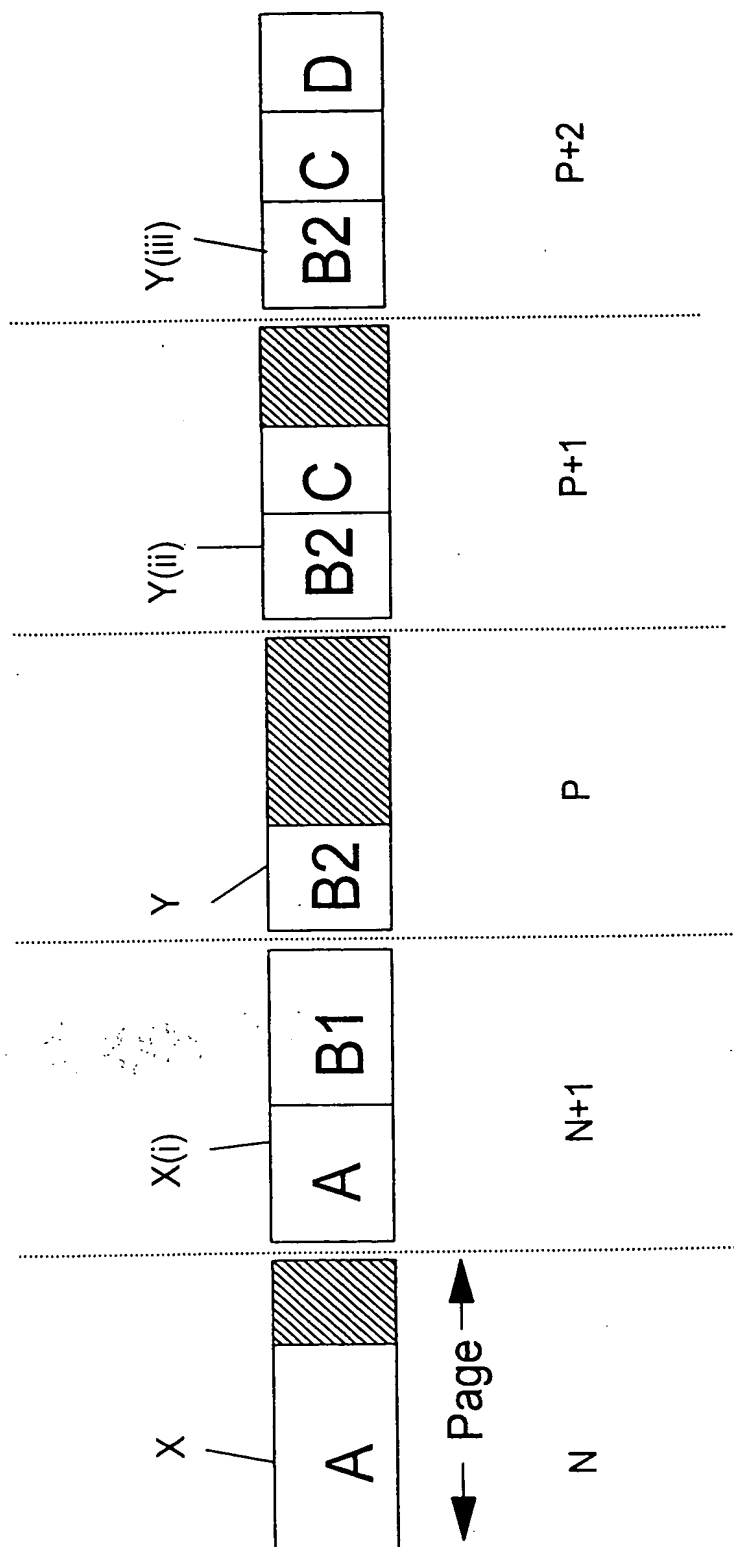


Figure 3

This Page Blank (uspto)

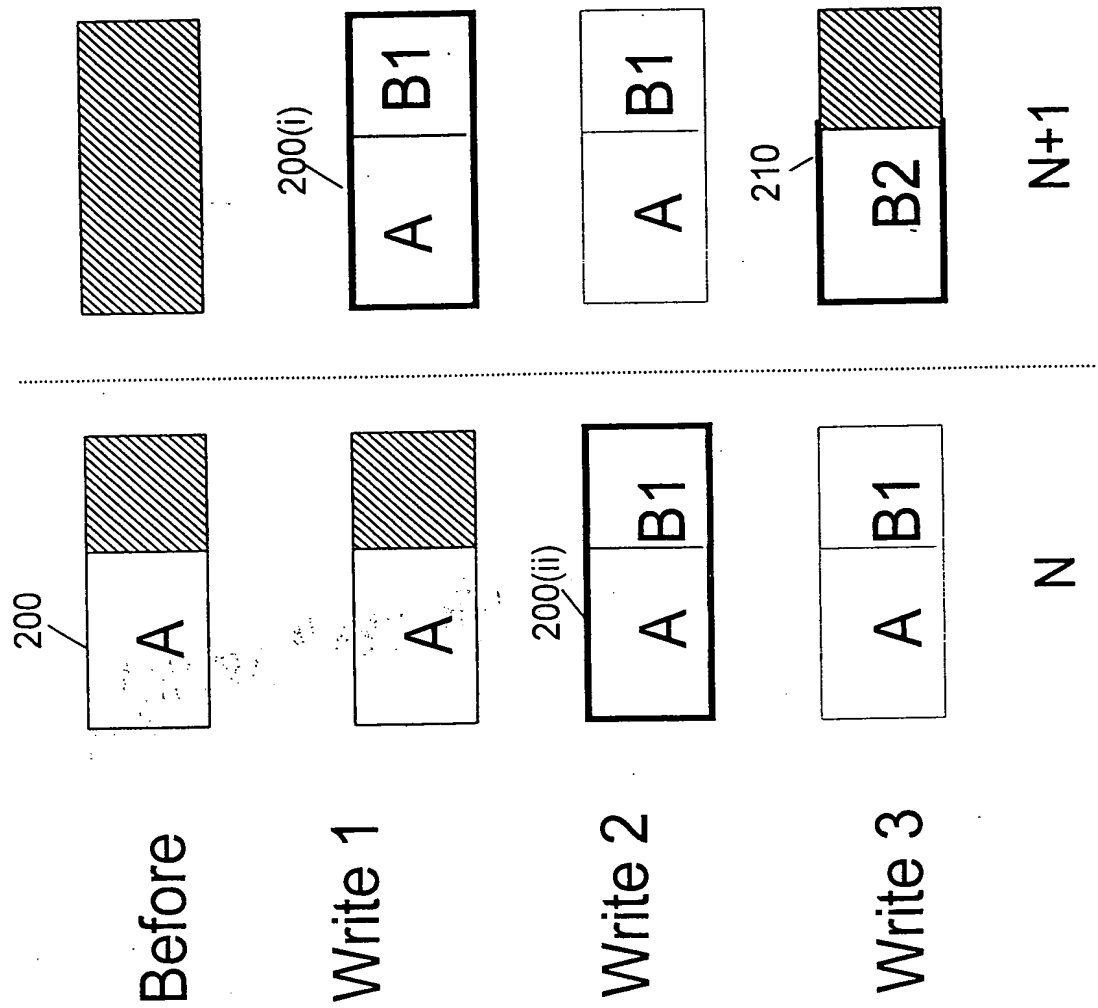


Figure 4

This Page Blank (uspto)

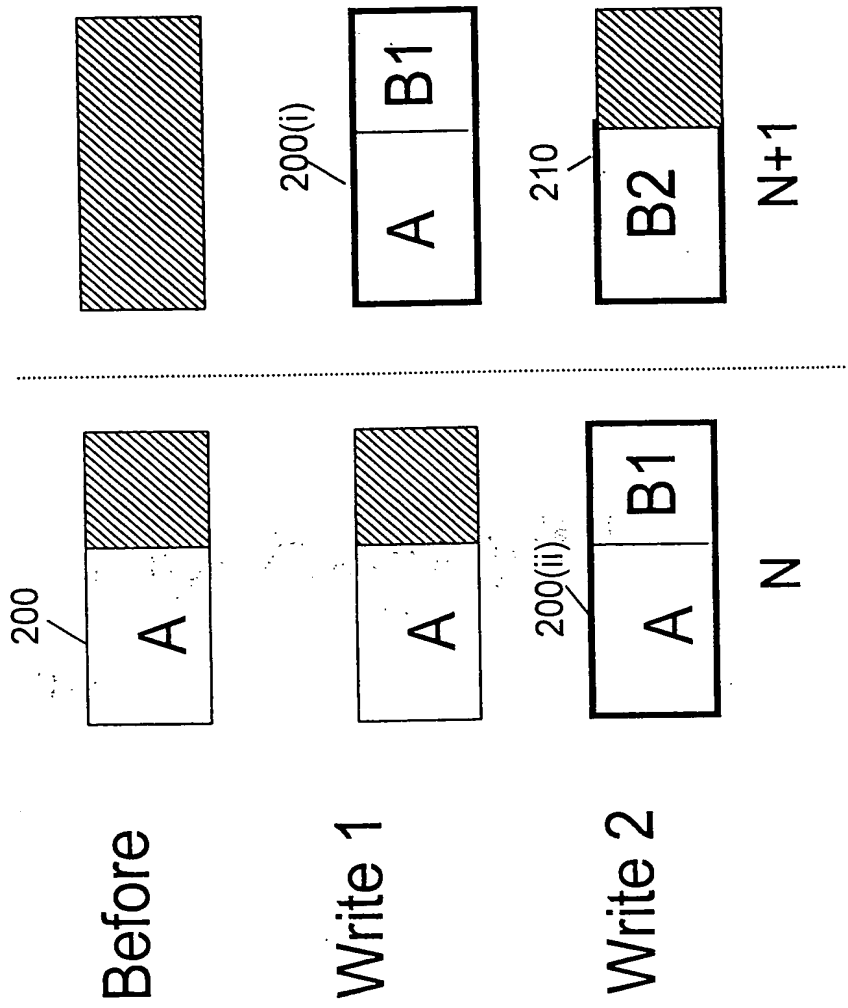


Figure 5

This Page Blank (uspto)

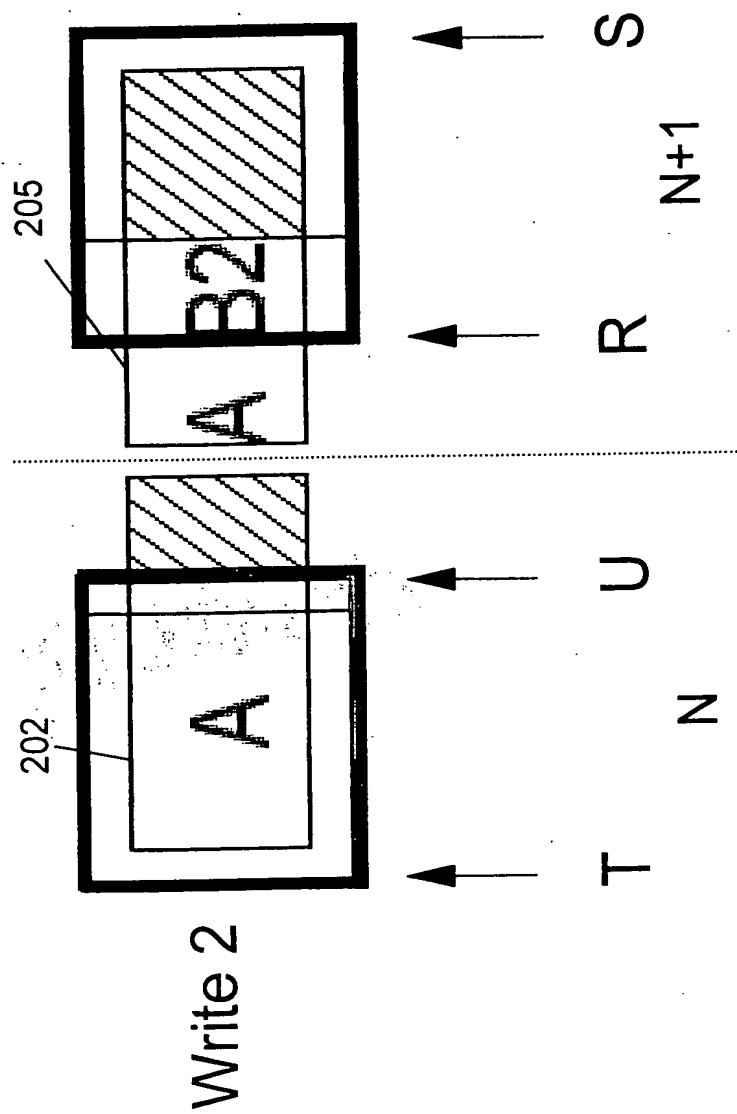


Figure 6

This Page Blank (uspto)

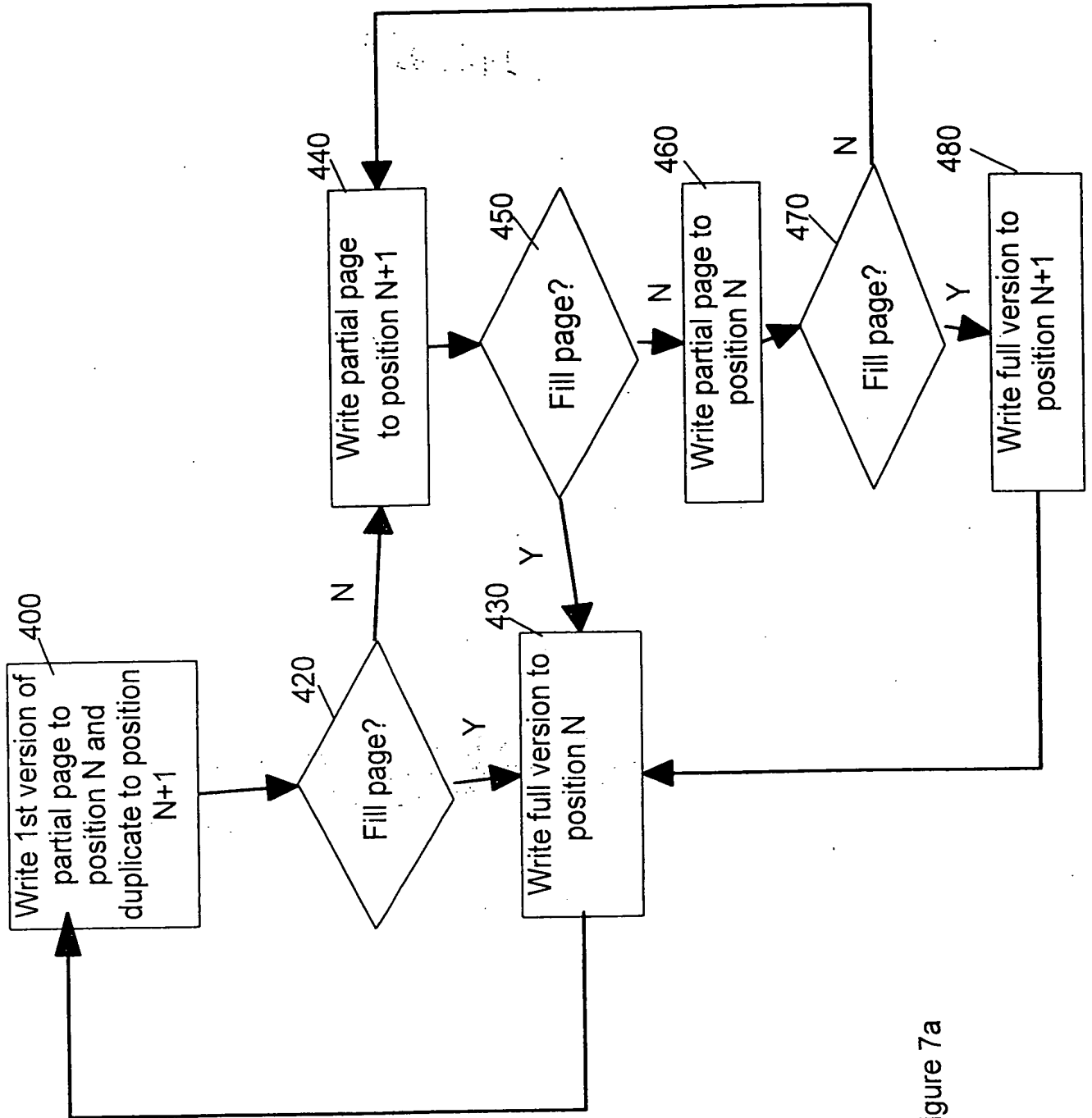


Figure 7a

This Page Blank (uspto)

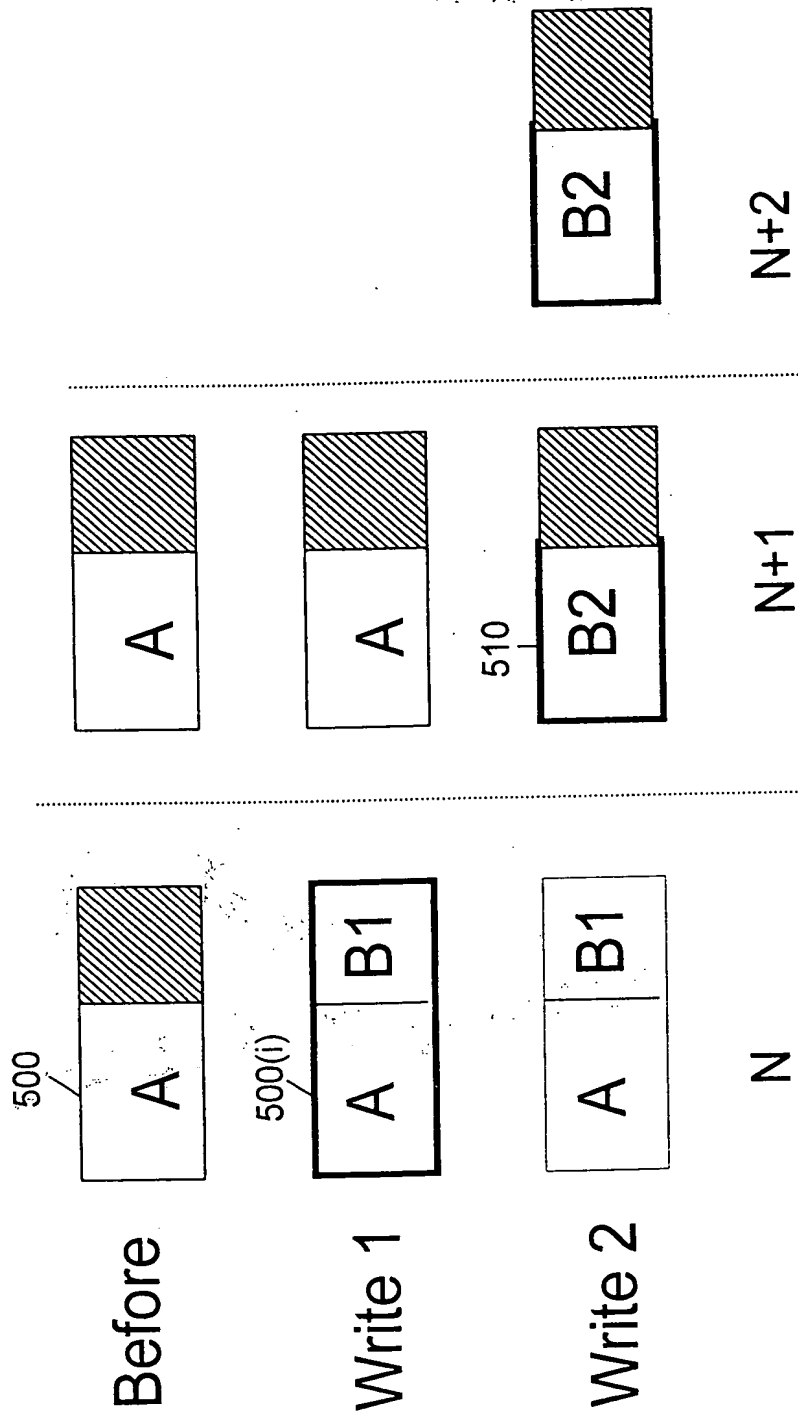


Figure 7b

This Page Blank (uspto)

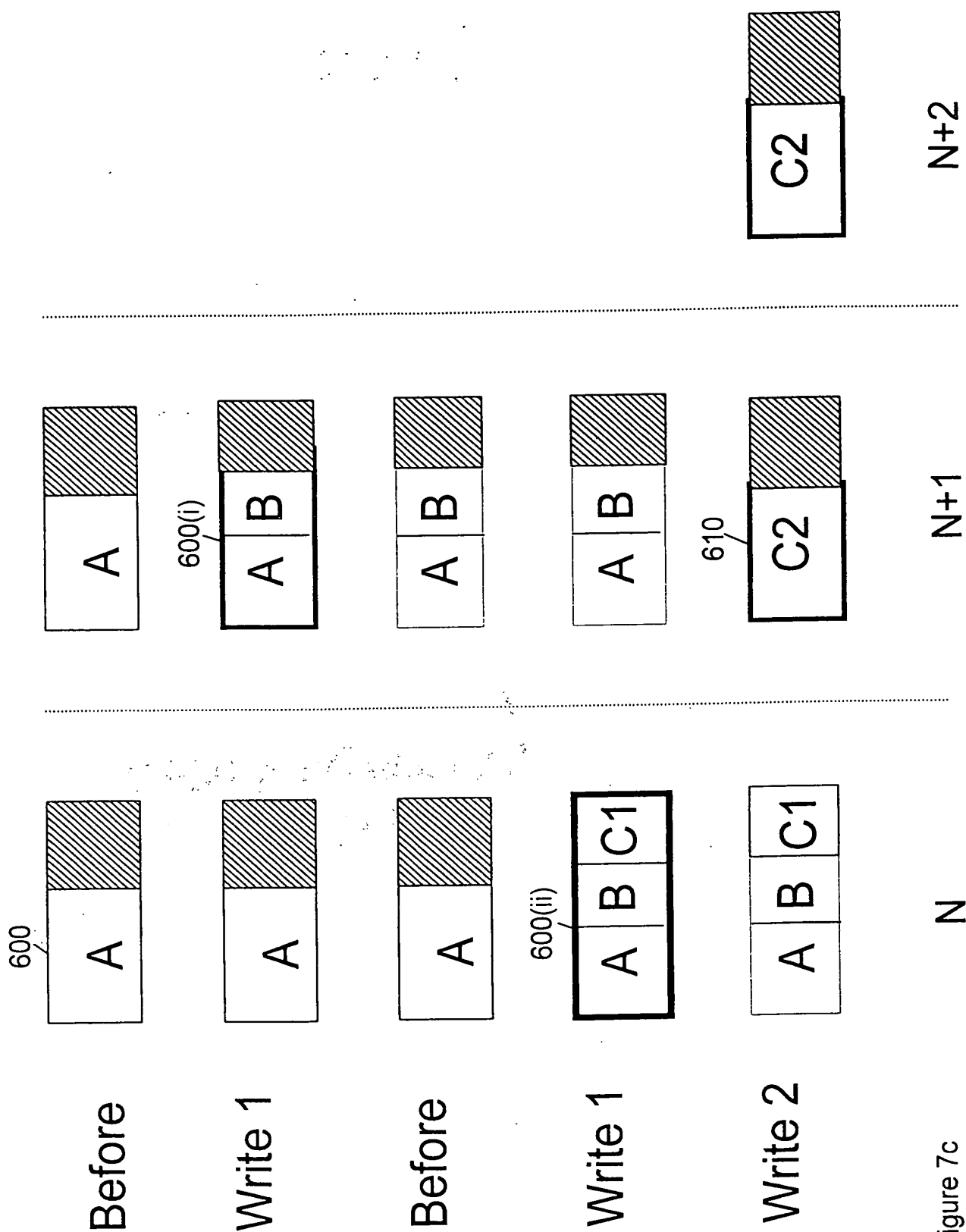


Figure 7c

This Page Blank (uspio)